TEAM PROJECT I:
'The Slaughterhouse Chronometer'

Final Product Report

Team Members
Bradley Coonan
Ye Ding
Changxun Li
Chris Rieger

Team Codename:
Slaughterhouse

Group #: 29

The University Of Queensland
AUSTRALIA

1.0 Table of Contents

2.0 Executive Summary 3
3.0 Project Solution 4
3.1 Mechanical Layout 4
3.2 User Manual 6
3.2.1 Powering the Clock 6
3.2.2 Software Installation 6
3.2.3 Graphical User Interface (GUI) Operation 6
3.2.4 Infrared (IR) Remote Operation* 7
3.3 Electronics Design 8
3.4 Embedded Firmware 9
3.5 PC – Based Software 12
3.6 Analysis of Final Product 13
4.0 Appendices 14
4.1 Appendix A: Bill of Materials and Project Budget 14
PCB manufacture budget: 17
4.2 Appendix B: Development Budget 18
4.3 Appendix C: Clock Schematics and PCB Design 19
4.3.1 PCB One 19
4.3.2 PCB Two 20
4.3.3 PCB Three 21
4.4 Appendix D: PCB Layouts 22
4.4.1 PCB One 22
4.4.2 PCB Two 23
4.4.3 PCB Three 24
4.5 Appendix E: AutoCAD Technical Drawings 25
4.5.1 Clock Top View 25
4.5.2 Clock Front View 26
4.5.3 Clock Side View 27
4.6 Appendix F: Firmware Flowcharts and Code Snippets 28
4.6.1 Main loop 28
4.6.2 Multiplexing and Infrared programming 29
4.6.3 Infrared button press parsing (handle_ir function) 30
4.6.4 Binary Lock Process 31

4.6.5 Binary Lock Code 32
4.6.6 LED Updating Code 33
4.6.7 GUI: Button Creation and Event Binding 34
4.6.8 GUI: Setting the Hours 34
4.6.9 GUI: Extracting Weather Data 34
4.6.10 Main Loop and ADC Code 35

2.0 Executive Summary

For the past thirteen weeks, Team 29 of ENGG2800 has been working on the design of a new digital clock with analogue display, as part of Project Clockwork Orange. In order to efficiently develop and implement designs pertaining to the manufacture of such a clock, the project was split up into three main subsystems: hardware, PC software and microcontroller software sectors. The final design results of each subsystem are as follows:

- Mechanical Layout: the main clock structure consists of an acrylic body with three PCBs attached via bolts, with separators along the shaft.
- Operation Manual: in order to communicate with the clock, optical communication systems have been implemented, including a standard remote control with modified button template, and a PC-based GUI capable of sending data (i.e. time, date and weather) through on-screen black/white flickering. Power is delivered to the clock via a power jack and three-way slide switch.
- Electronics: the main electronic components were selected based upon I/O pin requirements as well as relevant processor speeds and ease of implementation. Low cost was also taken into account and clock-face LEDs were chosen with PCB component density in mind.
- On-Board Software: the microcontroller software forms the basis for clock operation. The chosen microcontroller, ATmega168, has sufficient I/O pins to control all essential peripherals i.e. RTC, LED Driver, Transistors, LEDs and Light/IR sensors. The LEDs are controlled through layer multiplexing and shift registers.
- PC-Based Software: this software is utilised to set the time, alarm, and present weather data. These features will be implemented using an intuitive and aesthetic GUI.

Overall, each subsystem works seamlessly with the others, providing the clock with smooth operation and sound build quality. The final budget came to a low $79.449, which is well below the $100 maximum budget.

3.0 Project Solution

3.1 Mechanical Layout

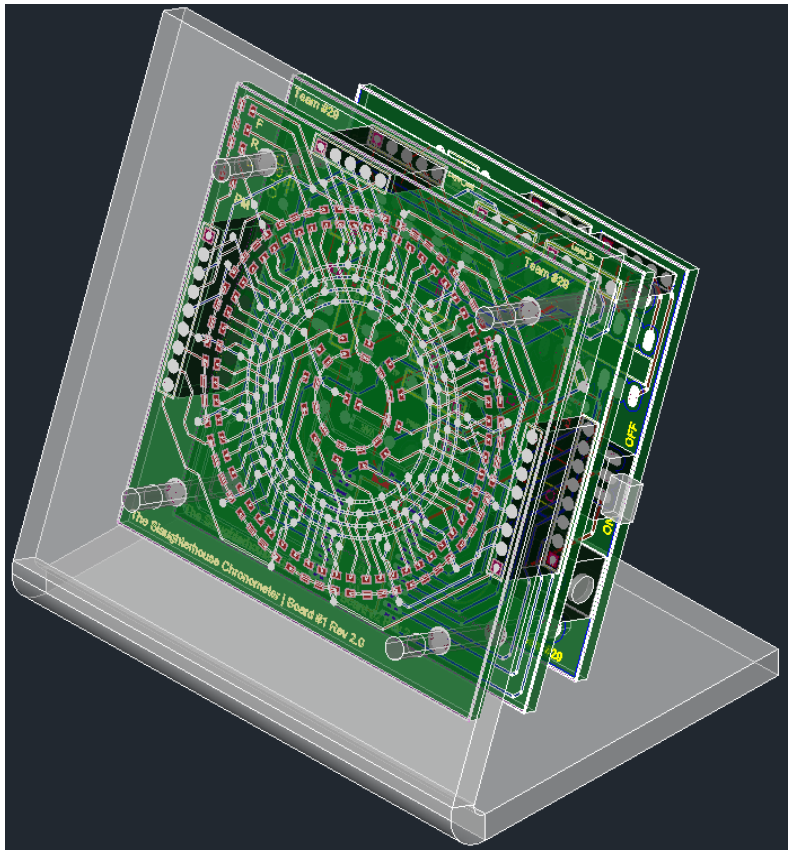


Figure 1: 3D Visualisation of the Final Product (Power Input and Switch are located on the Right-Hand Side of PCB 2) with Grey Acrylic for Ease of Viewing

The mechanical layout (Appendix 4.4 and 4.5) of the clock plays an integral part in the success of the final product, due to the compact nature and fragility of the electronic components. Without strong PCB mounts and intelligent placement of electronic components on the boards themselves, the feasibility of the project is significantly decreased. The integral parts of the mechanical system include the red acrylic body, screw mounts and the printed circuit boards (PCBs). The following section will look at how these parts are intertwined, and how the final design is benefitted as a result.

Fundamentally, the clock consists of three PCBs mounted upon a frame of red acrylic, with a 75mmx75mm front panel (Appendix 4.5.2). With the acrylic bent back at an 80 degree angle, the centre of mass is located just in front of the clock centre. This works to compensate for the weight of the PCBs and screws, and essentially prevents the clock from tipping forwards or backwards. In the event of the clock being bumped, having the centre of mass slightly forward is beneficial, as it increases the chance of the clock falling onto the acrylic panel rather than the delicate PCBs on the rear. Non-slip pads are also present on the acrylic's base, as a means of extra stability. The PCBs are mounted using 40mm M3 screws, with four screw holes pre-drilled in a 45mm x 45mm array (Appendix 4.5.2) for each PCB. Each screw will go through each PCB so that they are aligned correctly. This will provide excellent stability with regards to PCB movement; and with separators

placed at 12mm intervals along the shaft, will prevent unintentional contact between PCB components. Additionally, acrylic washers have been placed between the separators and PCBs, due to the possibility of PCB traces being conducted through the bolts to ground. The high rigidity of the acrylic panel will also assist in stabilising the screws. This is essential due to the minimal 4mm thickness of the acrylic.

The placement of the components on each of the PCBs is an important aspect to consider, as different components have higher or lower importance with regards to the operation of the electronic circuitry. The small footprint of the PCBs also restrict the location of components, therefore, smart component placement is vital.

The front PCB (Appendix 4.4.1) is a 60mm x 60mm board, contains the clock-face and is the most densely packed PCB out of the three. The concentric circles which represent the hour/minute LED layout provide the most readable clock-face and are arranged so as to maximise all available space. An ultra-bright LED is also positioned at the circle centre as a programming indicator. One problem of this design arose when it was realised that each LED would need to be addressed individually. In order to overcome this problem, the 72 hour/minute LEDs were split into five layers (Appendix 6.2 – Image 2), and multiplexing was utilised. The placement of weather and AM/PM LEDs in the top left corner also allows more room between the drill holes for the minute/hour LEDs to be arranged. In contrast to previous prototype designs, headers have been utilised in the final design in order to easily connect each LED trace and transistor layer to PCB Two; thus improving ease of debugging and part maintenance. The middle PCB (Appendix 4.4.2) is also a 60mm x 60mm board, and contains the LED driver, as well as the multiplexing transistors and op amp circuitry. These essential components are placed on the middle board due to their importance in interfacing with and controlling the clock-face LEDs. The transistors are grouped in the upper region of the PCB (Appendix 6.2 – Image 5); with the LED driver residing on the reverse side of the board, in order to compensate for the additional headers connecting to PCB One. The central placement of the op amp circuitry also allows easy access to the optical light sensor connections. Additionally, the optical and light sensors have been placed near the board edges in order to allow signals to be picked up uninterrupted. These main constituents of PCB Two are intelligently placed around a ground pin (Appendix 6.2 – PWR Header), which allows each component to be routed to a single ground. This board is also wire free, with headers acting as connectors to both adjacent boards.

The final PCB (Appendix 4.4.3) is the smallest board at 56mm x 56mm and houses the most important components of the clock: the microcontroller, the power supply and the Real Time Clock (RTC). The microcontroller has been placed on the third board as it allows ease of access to the two boards above it, via headers. These headers are located on either side of the microcontroller. The fact that the microcontroller is also the most manipulated component warrants its placement on the rear PCB, as it allows for easy detachment. The RTC is situated directly to the left of the microcontroller. PCB Three also includes a slide switch, battery cell and speaker. The power switch is located directly above the power jack, with the switch facing outwards to allow easy access. The battery holder, which powers the RTC, is situated on the rear of the PCB due to its large size; whilst the speaker is located along the bottom edge, adjacent to the power input (Appendix 4.4.3). Due to the high level of electronic activity on this PCB, a ground plane has been added in order to reduce the electrostatic noise of the circuitry.

3.2 User Manual

3.2.1 Powering the Clock

1. Connect the 2.1mm in-line adapter (7-12V) to the power jack located on the right hand side of board two.
2. Plug the other end of the power lead into a power outlet socket.
3. Move the slide switch onto the 'ON' position.

3.2.2 Software Installation

GUI

1. Open USB directory
2. Open 'GUI' Folder
3. Double-click 'LED_Clock_GUI_v5.exe'

3.2.3 Graphical User Interface (GUI) Operation

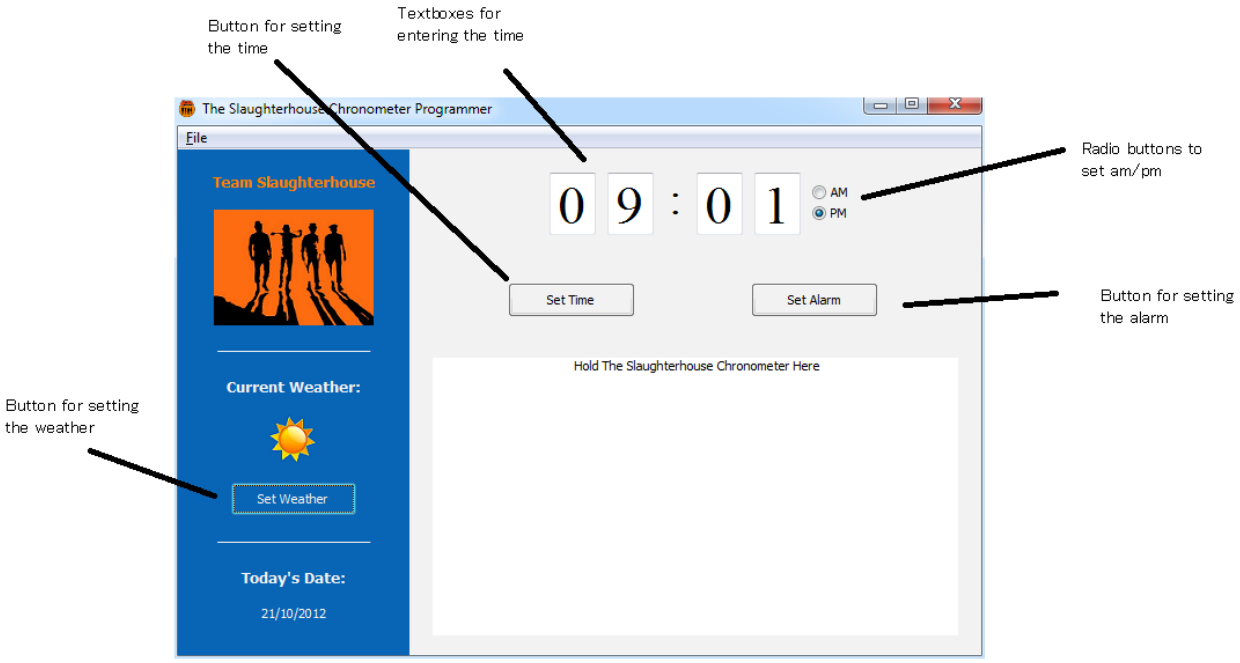


Figure 2: GUI for PC Software

Setting the Time

1. Use the textboxes to input the desired time: From the left, enter hours into the first two textboxes, and minutes into the last two textboxes.
2. Select an AM/PM radio button.
3. Hold the Slaughterhouse Chronometer with the light sensor (located on the left of the clock) facing the area displaying "Hold The Slaughterhouse Chronometer Here'.
4. Press Set Time

Setting the Alarm

1. Repeat steps 1-3 from Set Time.
2. Press Set Alarm

Setting the Weather

1. Hold The Slaughterhouse Chronometer with the light sensor (located on the left of the clock) facing the area displaying "Hold The Slaughterhouse Chronometer Here'.
2. Press Set Weather

3.2.4 Infrared (IR) Remote Operation*

*Setting the Time***

1. Aim the remote at the Slaughterhouse Chronometer and press Set Time.
2. Using the numerical buttons, enter the desired time as a 4-digit 24 hour time representation: The first two digits represent hours, and the final two digits represent minutes.
3. Press Set Time.

*Setting the Alarm***

1. Aim the remote at the Slaughterhouse Chronometer and press Set Alarm
2. Using the numerical buttons, enter the desired time as a 4-digit 24 hour time representation: The first two digits represent hours, and the final two digits represent minutes.
3. Press Set Alarm.

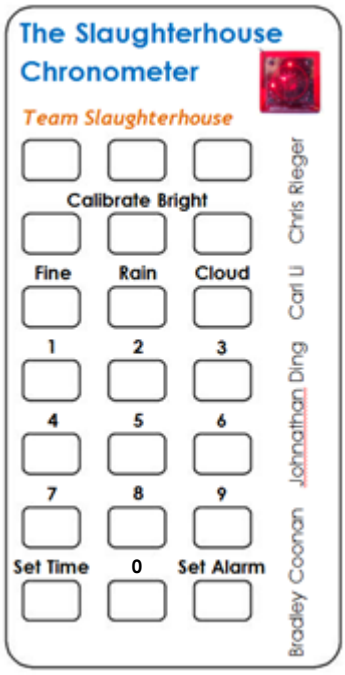


Figure 3: IR Remote Overlay

Setting the Weather

1. Press the desired weather condition button to set the weather: 'Fine', 'Rain' or 'Cloud'.

Calibrate Brightness

1. Position the Slaughterhouse Chronometer in an area of desired brightness.
2. Press Calibrate Bright.
3. The clock face will now be brightest when situated in an area of this brightness.

* Note that as long as the LED in the centre is on, only the Set Time/Set Alarm and the numeric buttons are processed. All other button presses are ignored. When not in programming mode, the Set Time/Set Alarm, the weather control and the calibrate bright buttons are active.
** If less than four digits are entered, the operation is ignored. If more than four digits are entered, only the first four are used.

3.3 Electronics Design

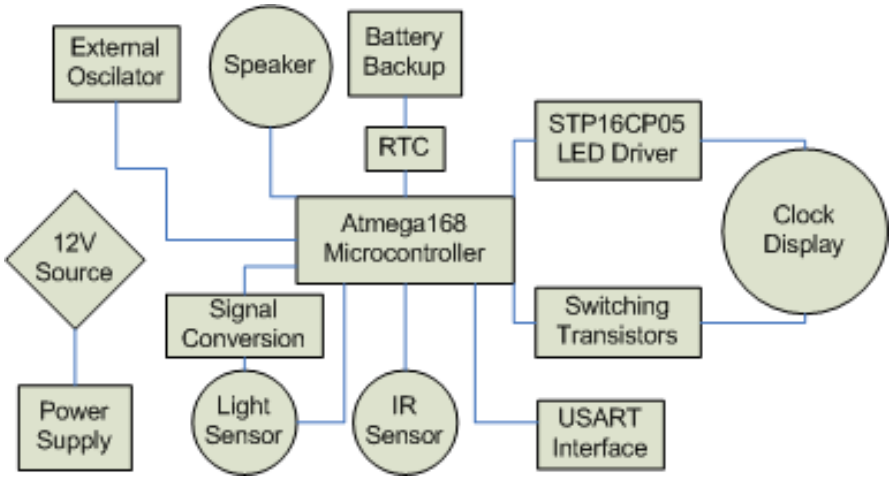


Figure 4: Electronics Subsystems Flowchart

The electronic design plays a key role due to the large number of peripherals that are interfaced with, as well as having both analogue and digital measurement circuitry. The above diagram gives an overview of the electronic interfacing required. The schematics (Appendix 4.3) have been split into three key sections corresponding to the three individual PCB's needed.

The front of the board consists of 72 LEDs for time display with an additional 3 LEDs for weather, 1 LED for AM/PM display and 1 system LED. Surface mount, 0805 package, red LEDs were chosen in order to meet the tight board dimensions, whilst still being moderately easy to solder. These LEDs have 180mcd output power each when driven at full current. The chosen LEDs have reasonably bright illumination, with the colour chosen to best illuminate through the given red acrylic.

The large number of LEDs are controlled primarily through a current controlled 16 bit shift register; the STP16CP05. The shift register's current controlling ability makes it very suitable for this application in order to keep brightness levels of LEDs relatively the same. As seen in the schematic for the front PCB, it is seen that the LEDs are arranged in 5 distinct layers of 15 LEDs. This results in only 1 of 5 layers being turned on at a time, relying on the inertia of vision and fast multiplexing to allow for the addressing of LEDs simultaneously on varying layers. The current controller has been set to around 20mA per output line (keeping in mind a 20% duty cycle), which is set using a resistor (R-ext.) of 1KΩ, as per the datasheet.

The STP16CP05 has an enable line that is sent PWM signals to adjust the brightness level of the LEDs, a key feature of the product specification. This eases the MCU load from adjusting duty cycles through firmware. The layers are switched using BC337 NPN transistors, chosen for their low cost, high collector current (0.8A) capability, and fast switching functionality. As seen in the schematic (Appendix 4.3.2) the transistor emitter is current controlled by the driver, doing away with the need of a current limiting resistor on the MCU output pins.

The microcontroller selected is the ATmega168, which was based on a number of factors, including low price, large level of input/output pins, and specification adherence. This microcontroller gives

reasonable flash/ram size with the code release using approximately 40% of the available flash. A 16 MHz external clock has been included allowing for greater cycles/sec.

The PCF8563 is the RTC (Real Time Clock) used in the clock, based on its low price and adequate feature set. It communicates to the ATmega168 through the I2C bus and is battery powered from a single 3V lithium cell. The RTC features a calendar and alarm functionality. A 32.768 kHz crystal is used alongside the clock as specified in the datasheet. Pull-up resistors have been added to the SDA and SCL lines in order for the open-drain lines to function.

A standard photo resistor mounted on the middle PCB is used for both brightness detection and optical communication from the PC. The photo resistor is in a voltage divider configuration with the output given to an op-amp acting as a voltage buffer. This voltage is read directly from the ATmega168's on-board ADC. It is also fed into another op-amp acting as a comparator in an open-loop configuration, resulting in infinite gain bounded by the operating voltages. The comparator compares the photo resistor's output voltage to a variable set point, adjustable through a potentiometer. This allows the brightness threshold to be adjusted as needed for the programming mode. A three-pin TSOP4038 IR receiver with built in filtration and signal amplification was used. The output from this IC is simply connected to a digital I/O pin.

A 5V 7805 voltage regulator with up to 800mA current draw was selected for this project. A schematic with this regulator and capacitors for noise filtering was followed resulting in a clean voltage source with minimal ripple. A diode was added on the input side to mitigate any potential reverse polarity issues. The power consumption of the clock can be reduced by activating either software dimming (via IR remote) or adjusting the current to the STP16CP05 through the external resistor, decreasing the output current to the LEDs. Decoupling capacitors were added on every board near the IC's to reduce any power related issues which may be introduced from fast power switching carried out by the transistors.

A piezo element was added which is attached to a PWM output line used for alarm functionality. A spare I/O pin is available at a header as well as USART TX/RX lines which allow for debugging if required.

3.4 Embedded Firmware

The embedded firmware also takes on a very modular approach, grouped into the following components, organised in 3 '.h' files and 2 '.c' files: orange.h, orange.c, rtc.h, rtc.c and ir.h. A list of the code's modules and their file dependencies is as follows:

- Main loop – RTC polling, alarm checking, auto-dimming using ADC (orange.h, orange.c)
- Alarm sounding (rtc.h, rtc.c, orange.h, orange.c)
- LED driver, transistors and ADC for LED multiplexing and dimming (orange.h, orange.c)
- RTC interfacing via the I2C bus – reading/writing time and alarm (rtc.h, rtc.c, orange.c)
- Infrared remote programming – setting time, alarm, weather and calibrating brightness (orange.c, ir.h)
- Optical programming – setting time, alarm and weather (orange.h, orange.c)

Main loop

The main loop (Appendix 4.6.1), an infinite loop, continuously polls the RTC using the functions in rtc.c (to be explained). Upon polling the RTC, with every second, the display of the clock face is updated.

After this, the current time and the stored alarm time (which are both saved between microcontroller resets due to the presence of the RTC) are checked. If they match, the alarm is sounded with every iteration until 10 seconds have elapsed, and then it is turned off.

Lastly, the main loop initiates an ADC conversion (Appendix 4.6.3) of the raw output from the LDR, to obtain a value that is used as a scaling factor for a PWM duty cycle to control the brightness of the LEDs. A value from 0 to 170 (by default, but can be changed at runtime) is scaled to a value from 255 to 0 (active low device). Appendix 4.6.10 shows the code for the ADC conversion.

Alarm sounding

The piezo speaker used for the alarm is driven by a PWM signal using timer 0 on the microcontroller. The alarm sounds for 10 seconds with the pitch of the tone changing between two distinct tones every second.

LED driver, transistors and ADC for LED multiplexing and dimming (Appendix 4.6.2)

To turn on a particular LED requires that a particular transistor be turned on and the correct bit pattern shifted into the LED driver. The *update_seconds, update_minutes, update_hours,* and *update_weather* are auxiliary functions that generate the corresponding bit pattern for the relevant transistor layer to output to the driver.

For example, the hours and the weather display pattern is enabled by the fifth transistor, and hence share the same bit pattern, and is of the following format, in large-endian format:

14	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Fine	Rain	Cloud	11	10	9	8	7	6	5	4	3	2	1	0 (12)

The bit pattern is hence, simply 1 bitwise shifted left *n* times, where *n* is the hours value. When the weather is set, the pattern is bitwise OR'd with the existing pattern, and displayed on the LEDs whenever the fifth transistor is on.

With the minutes and seconds LEDs, they are arranged in a fashion such that every group four is connected to a driver pin, and within the group, connected to each of the four transistors. The format is as follows, where the rows correspond to transistors 1, 2, 3 and 4, and the driver pins also in large-endian format:

	14	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	56	52	48	44	40	36	32	28	24	20	16	12	8	4	0
1	57	53	49	45	41	37	33	29	25	21	17	13	9	5	1
2	58	54	50	46	42	38	34	30	26	22	18	14	10	6	2
3	59	55	51	47	43	39	35	31	27	23	19	15	11	7	3

In order to light up a particular LED, corresponding to either minutes or seconds x: firstly the transistor t that is required to be turned on is given by t = x mod 4, and secondly the driver pin p that is required to be set to 1 is given by p = x / 4. Knowing this, the functions can generate the correct pattern for the correct transistor. In addition to this step, *update_seconds* also preserves the current minutes value using a bitwise OR. *update_minutes* also sets the first bit of the pattern for the 0th transistor as 1, since whenever a minute ticks, the seconds value must be 0. Appendix 4.6.6 shows the four LED updating functions.

Every 800µs, triggered by a timer overflow interrupt combined with a timer prescale, the current transistor is turned off, the pattern is shifted into the LED driver, and lastly the next transistor is turned on.

RTC interfacing via the I2C bus

For interfacing with the RTC, the I2C-compatible TWI interface on-board the AVR was used. Code was sourced from an online project, http://www.ermicro.com/blog/?p=950, and modified for use in this project (acknowledged and referenced in-code).

The functions provided were the very basic I2C interfacing functions: start/stop condition and read/write instructions. Using a combination of these, more complex functions were built to read and write the time and alarm. For time read/write, the initial starting address is at 0x02, and 0x09 for alarm read/write.

For reading, the process is as follows:

1. Start condition
2. Write starting address
3. Stop condition
4. Start condition
5. Read instruction(s)

⋮

n. Stop condition

And for writing, the process is as follows:

1. Start condition
2. Write starting address
3. Write instruction(s)

⋮

n. Stop condition

Infrared remote programming

The infrared interface (Appendix 4.6.3) operates on the same interrupt as the multiplexing. In fact, the period of multiplexing was chosen based on the 1.6ms delay between the bits of the waveform sent out by the infrared remote.

Every time the interrupt service routine runs, it checks the output of the infrared receiver for a low signal. If that is the case, then values on that pin will be recorded. From then on, each firing of the interrupt will store the current bit of the infrared waveform. Once the 8 bits are received, the waveform is processed by the *handle_ir* function. Within this function, the waveform received is compared with pre-recorded values and then relevant tasks are performed.

The first Set time/Set alarm button press triggers the device to be in programming mode. The first four numeric button presses from then are stored. Any further presses will be ignored, as is the case

for any non-numeric buttons. Pressing Set time/Set alarm button again will exit programming mode and write the time or alarm.

There are also 3 buttons on the remote that allows the user to easily set the weather to fine, rainy or cloudy. There is also a calibrate brightness button – this button reads the current ADC value and sets it as the upper threshold of the ADC bound value, so that the current ambient light level is registered at 100% brightness. The hardcoded value for this is, as previously mentioned, 170.0, and this is a very average, typical laboratory light level. Having a brightness calibration feature is useful in situations where there is extremely strong or weak ambient lighting, but dimming is still needed.

Optical programming

External interrupts acting on pin change and timers were used from communications between the clock and the PC software. This is done by measuring the amount of time that has been passed from the last interrupt to find how many bits that has been received. For example if a bit is to be received every 100ms and the time between when the last interrupt occurred is 230ms, two bits would have been received. As the external interrupts work on pin change the two bits that were received is the opposite of the value that is currently read on the pin connected to the light sensor.

Before the clock is able to record any data, a mechanism similar to a 4 bit binary lock is used. This is used to identify that the data received is from the PC software to prevent data from been overwritten unintentionally due to changes in luminosity from the environment. Furthermore two more bits are to be matched after receiving the data from the PC software to verify that no errors has occurred in between. The code for the binary lock is shown in Appendix 4.6.5.

3.5 PC – Based Software

The PC software is made up of three modules, the GUI for interacting with the user, the module for communicating with the clock and the module for retrieving data from the weather server. How these modules interact with each other is shown in Figure 5 to the right.

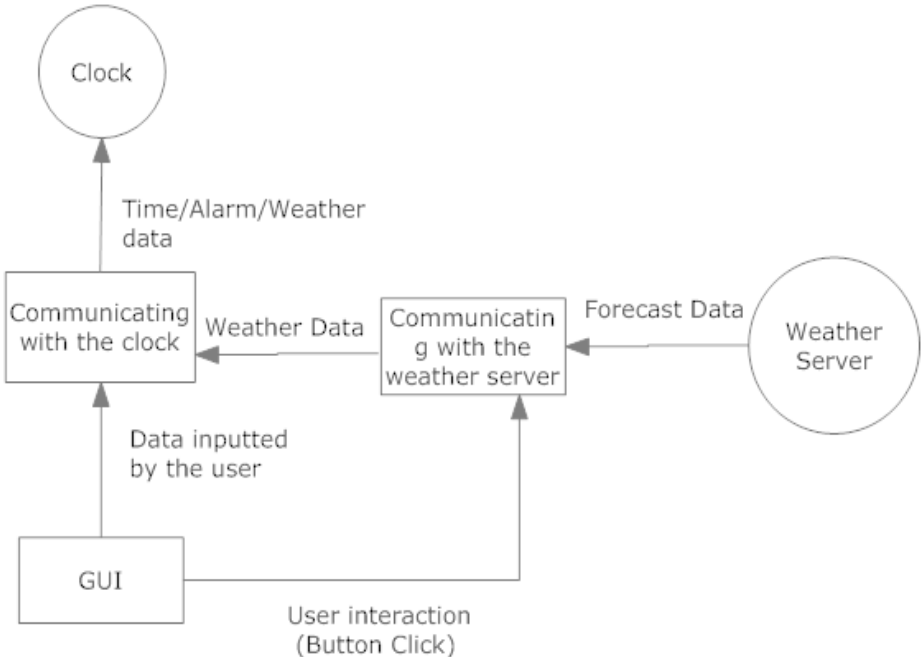


Figure5: PC Module Flowchart

The GUI for interacting with the user mainly consists of code that specifies the layout of the components (Buttons, text boxes, frames etc.) and binding events (Button press, key press etc.) associated with these components. The code in Appendix 4.6.7 performs the task of laying out the set time button and binding the necessary function to it.

The module for communicating with the clock consists of grabbing the information from the GUI, converting the data into binary then sending the binary data to the clock through a series of black and white flashes. As described in the firmware section, there are four bits at the start that must be sent to indicate the data is coming from the PC software, two bits after that to indicate its mode of operation (setting the time/alarm/weather) and two more at the end for error checking. The code in Appendix 4.6.8 shows the necessary operations required to send the hours part of the time to the clock.

The module for setting the weather is done by first retrieving the data from the weather server, parsing the data given, then giving the data to the module that communicates with the clock. The code in Appendix 4.6.9 shows the steps for retrieving and parsing the data from the weather server.

3.6 Analysis of Final Product

When combined, these subsystems form a viable, seamless and accurate piece of timekeeping technology. The hardware was designed with simplicity in mind, and works perfectly with the developed firmware in order to provide an intuitive and aesthetically pleasing product. Additionally, optical and IR communication with the device is smooth and user-friendly. An intuitive GUI and a logical IR remote layout further promote the ease of use of the clock. Currently, the final design obeys all rules set in the initial project specifications; however, this is the result of different successes and failures throughout the project timeline.

The main success for the project was getting a batch of PCBs sent away within a few weeks of receiving the task. This allowed prototyping to begin quickly, which in turn improved the build quality of each subsequent PCB set. Decisive actions from the team are the main reason this could be achieved, with an almost constant stream of communication between team members. Whilst most of the project progressed smoothly, certain failures led to short setbacks in the prototyping phase. Notably, the first PCB run had various errors with regards to hole sizes, pin alignment and component placement. The incorrect positioning of a PCB trace resulted in a microcontroller pin being connected to the wrong component. This failure, in turn, caused a setback in IR testing. Whilst these errors were present, the team managed by directly modifying the boards to allow prototyping to continue. In the final PCB batch these errors were all rectified, however, extra discrepancies were still found. These minor errors mainly affected the physical side of the design - such as preventing a nut being screwed onto a bolt – and therefore were easily solved.

Overall, the final product works efficiently and complies with the product specification, with all errors along the way being resolved.

4.0 Appendices

4.1 Appendix A: Bill of Materials and Project Budget

The following table summarises the parts decided upon for the project. Note that some parts have a minimum order quantity, resulting in more components listed than needed.

Part Description	Model	Supplier	Order Number	Link	Price
LEDS	0805 SMD	Element14	2099241	http://au.element14.com/kingbright/kpt-2012surck/led-red-0805-smd/dp/2099241	100 x $0.11 = $11
LED Driver	STP16CP05MTR SMD	Element14	1542235	http://au.element14.com/stmicroelectronics/stp16cp05mtr/ic-sm-driver-led-sink-16-bit/dp/1542235	1 x $4.21 = $4.21
Real Time Clock	PCF8563	Element14	2064559	http://au.element14.com/nxp/pcf8563p-f4/ic-rtc-calendar-8563-dip8/dp/2064559	1 x $4.12 = $4.12
32.768khz Crystal	AB38T	Element14	1611828	http://au.element14.com/abracon/ab38t-32-768khz/crystal-32-768k-12-5pf-cl-watch/dp/1611828	1 x 0.42 = $0.42
Piezo	PKM13EPYH4000	Element14	1192513	http://au.element14.com/murata/pkm13epyh4000-a0/sounder-4khz-12-6mm/dp/1192513	1 x $1.34 = $1.34
Voltage Regulator	MC78M05CDTRKG	RS Components	688-9294	http://australia.rs-online.com/web/p/linear-voltage-regulators/6889294/	2 x $0.47 = $0.94
Transistors NPN	BC33725TA	RS Components	671-1116	http://australia.rs-online.com/web/p/bipolar-transistors/6711116/	10 x $0.08 = $0.80

20mm 3v coin cell	CR2032	Element14	2065171	http://au.element14.com/multicomp/cr2032/coin-cell-lithium-3v-210mah-cr2032/dp/2065171	1 x $0.75 = $0.75
Battery Holder	CH25-2032LF	Element14	2064715	http://au.element14.com/multicomp/ch25-2032lf/battery-holder-dip-20mm/dp/2064715	1 x $0.54 = $0.54
Op-Amp	LM358P	Element14	1648686	http://au.element14.com/texas-instruments/lm358p/ic-op-amp-dual-dip8-358/dp/1648686	1 x $0.34 = $0.34
Power Switch	MCLSS22	Element14	1522046	http://au.element14.com/multicomp/mclss22/slide-switch-dpdt/dp/1522046	1 x $0.707 = $0.707
DIP Sockets (Cut-able)	A 08-LC-TT	RS Components	674-2435	http://australia.rs-online.com/web/p/connector-ic-sockets/6742435/	10 x $0.09 = $0.90
Microcontroller	ATmega168P-20PU	RS Components	738-0375	http://australia.rs-online.com/web/p/microcontrollers/7380375/	1 x $6.50 = $6.50
Photo Resistor	VT935G	Element14	1652638	http://au.element14.com/excelitas-tech/vt935g/power-rating-80mw/dp/1652638	1 x $0.93 = $0.93
IR diode	TSOP4038	Element14	1798516	http://au.element14.com/vishay-semiconductor/tsop4038/ir-receiver-30m-950nm-through-hole/dp/1798516	1 x $1.24 = $1.24
Power Jack	MJ-179PH	Element14	1737246	http://au.element14.com/multicomp/mj-179ph/socket-low-voltage/dp/1737246	1 x $1.18 = $1.18
1.2k resistors	0W25 / 1K2 / 0.25 Watt / 5%	ETSG	14-20-08	http://itee.uq.edu.au/~etsg/	3 x $0.05 = $0.15

10k resistors	0W25 / 10K / 0.25 Watt / 5%	ETSG	14-21-09	http://itee.uq.edu.au/~etsg/	5 x $0.05 = $0.25
27pF Capacitor	Ceramic / 0805 / 27 / 27p / 200V	ETSG	64-02-06	http://itee.uq.edu.au/~etsg/	2 x $0.15 = $0.30
100nF Capacitor	Ceramic / 0805 / 100000 / 100n / 50V / X7R	ETSG	64-04-12	http://au.element14.com/murata/grm216r71h223ka01d/capacitor-0805-22nf-50v/dp/8820104	4 x $0.02 = $0.08
16Mhz Crystal	AS-16.000-18	Element14	1611761	http://au.element14.com/raltron/as-16-000-18/frequency-16mhz/dp/1611761	1 x $0.63 = $0.63
Potentiometer	CT6V202N	Element14	1227556	http://au.element14.com/te-connectivity-citec/ct6v202n/trimmer-2k2/dp/1227556	1 x $0.408= $0.408
Diode	1N4001-E3/73	RS Components	628-8931	http://australia.rs-online.com/web/p/rectifier-schottky-diodes/6288931/	10 x $0.07 = $0.70
Male Headers	CONNECTOR : PCB HDR - Single Row Header / Male / Straight / 1 x 40 / Gold Plated	ETSG	15-12-01	http://itee.uq.edu.au/~etsg/	1 x $0.60 = $0.60
Female Headers	CONNECTOR : PCB HDR - Single Row Header / Female / Straight / 1 x 40 / Gold Plated	ETSG	15-12-05	http://itee.uq.edu.au/~etsg/	1 x $2.55 = $2.55
Bolts	Screw / Pan / M3.0 x 30 / Zinc Plated Steel Supadrive	ETSG 10	10-58-01	http://itee.uq.edu.au/~etsg/	4 x $0.07 = $0.28

Spacers M3x10mm	Spacer / Female - Female / Brass Nickel Plated / Hexagonal / M3 x 10	ETSG	10-42-03	http://itee.uq.edu.au/~etsg/	6 x $0.35 = $2.10
Washers M3	Washer / Plain / M3.0 / Zinc Plated Steel	ETSG	10-70-02	http://itee.uq.edu.au/~etsg/	12 x $0.02 = $0.24
TOTAL					**$ 44.205**

PCB manufacture budget:

PCB unit cost is $2.20/sq. Inch.

PCB #	Dimensions	Cost
1	60x60mm = 5.58 sq. Inch	12.276
2	60x60mm = 5.58 sq. Inch	12.276
3	56x56mm = 4.86 sq. Inch	10.692
	Total	**$35.244**

TOTAL:

$79.449

4.2 Appendix B: Development Budget

Part Description	Model	Supplier	Order Number	Link	Price
M3x25 Bolt	Screw / Countersunk / M3.0 x 25 / Zinc Plated Steel Pozidriv	ETSG	10-50-02	http://itee.uq.edu.au/~etsg/	1 x $0.07 = $0.07
M3x20 Bolt	FASTENER : Screw / Countersunk / M3.0 x 20 / Zinc Plated Steel Supadriv	ETSG	10-50-01	http://itee.uq.edu.au/~etsg/	1 x $0.06 = $0.06
Solder	SOLDERING : Solder / Pb / 60% Sn, 40% Pb / 0,71mm / MP=188°C (Hobby Pack)	ETSG	A1-02-07	http://itee.uq.edu.au/~etsg/	5 x $1.00 = $5.00
Connector	CONNECTOR : Data / Plug / RJ45 / 8P8C / Austel A93-PL-0086	ETSG	15-10-03	http://itee.uq.edu.au/~etsg/	5 x $0.60 = $3.00
Zener 3v3	DIODE : 79 / Zener / 500mW / 3V3 / BZX79 Series	ETSG	04-58-01	http://itee.uq.edu.au/~etsg/	10 x $0.33 = $3.30
Zener 3v6	DIODE : 79 / Zener / 500mW / 3V6 / BZX79 Series	ETSG	04-58-02	http://itee.uq.edu.au/~etsg/	6 x $0.05 = $0.30

Total Development Budget = $11.73

4.3 Appendix C: Clock Schematics and PCB Design

4.3.1 PCB One

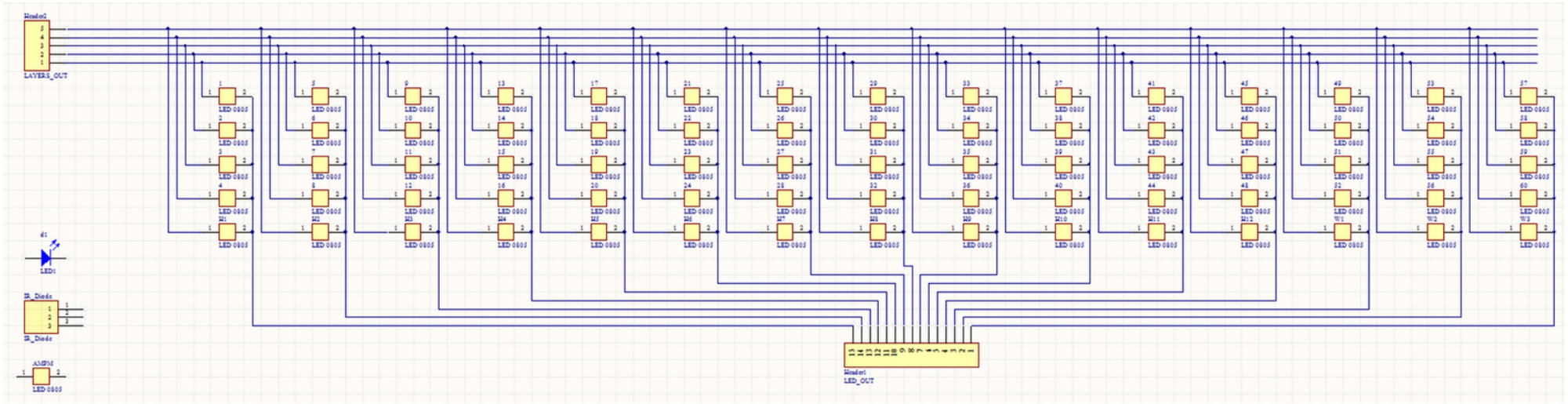


Figure 6: Circuit Schematic for PCB One

4.3.2 PCB Two

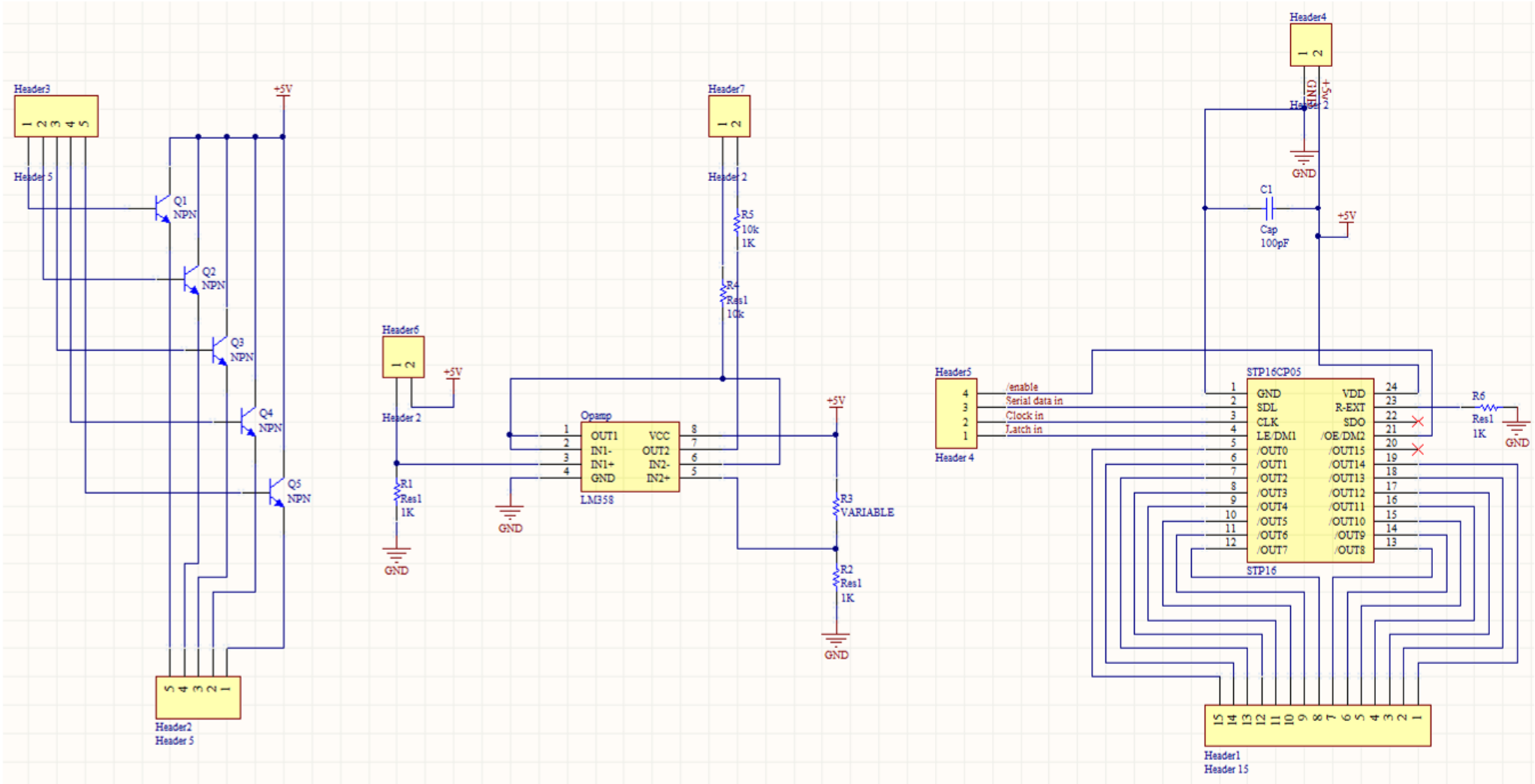


Figure7: Circuit Schematic for PCB Two

4.3.3 PCB Three

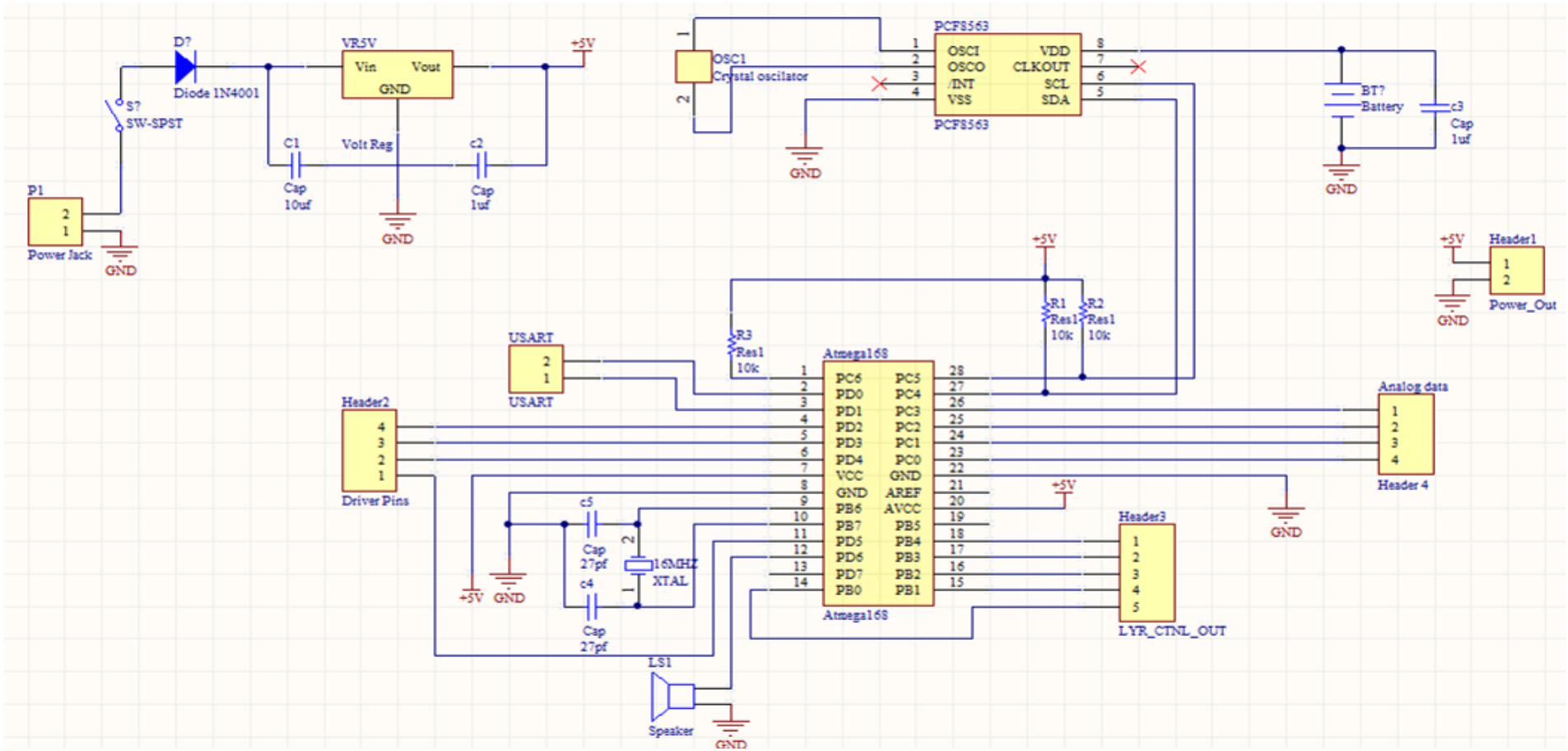


Figure 8: Circuit Schematic for PCB Three

4.4 Appendix D: PCB Layouts

4.4.1 PCB One

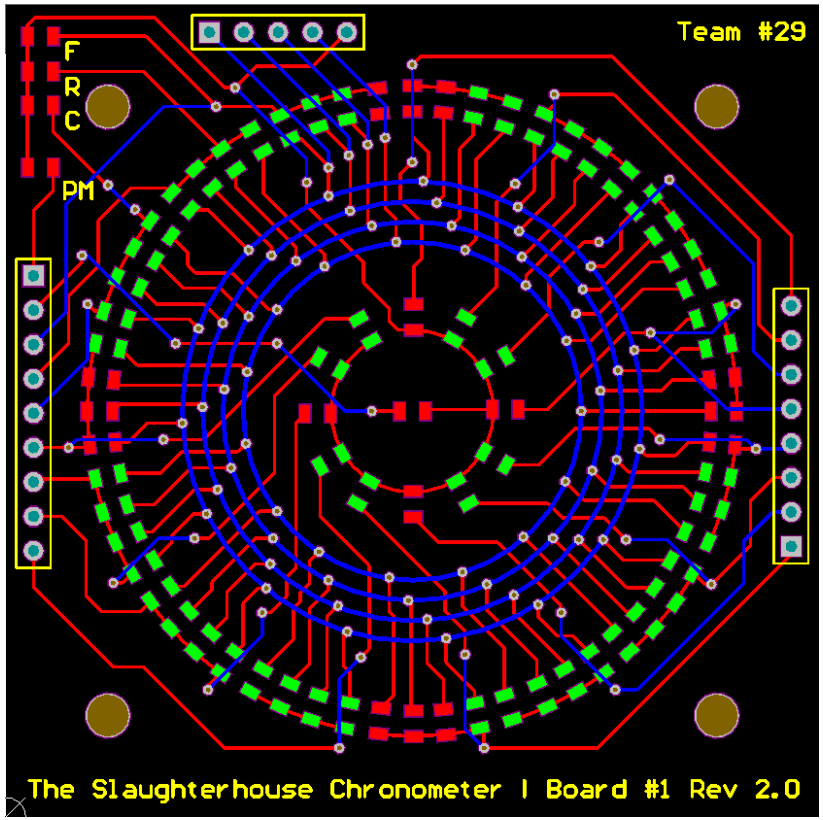


Figure 9: Front PCB Altium

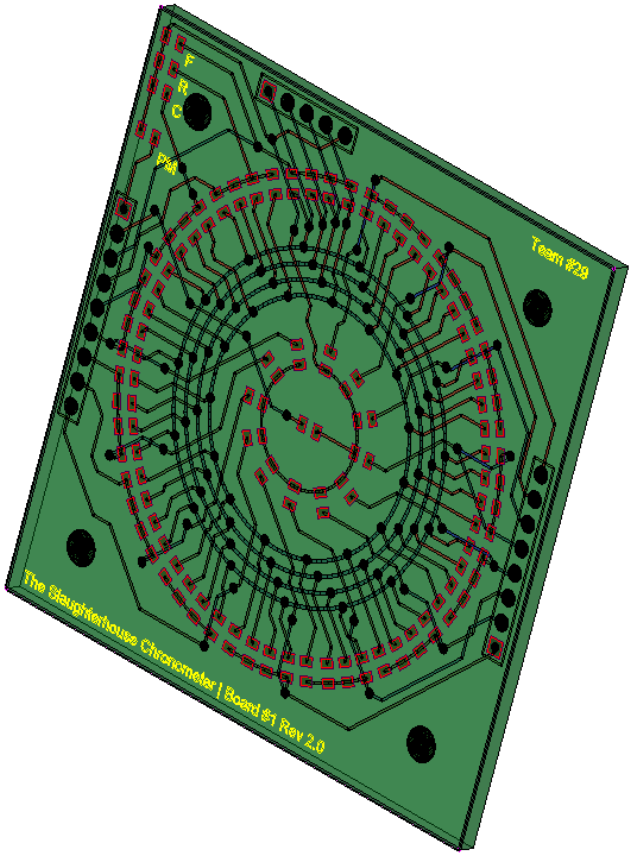


Figure 10: Front PCB 3D Visualisation

4.4.2 PCB Two

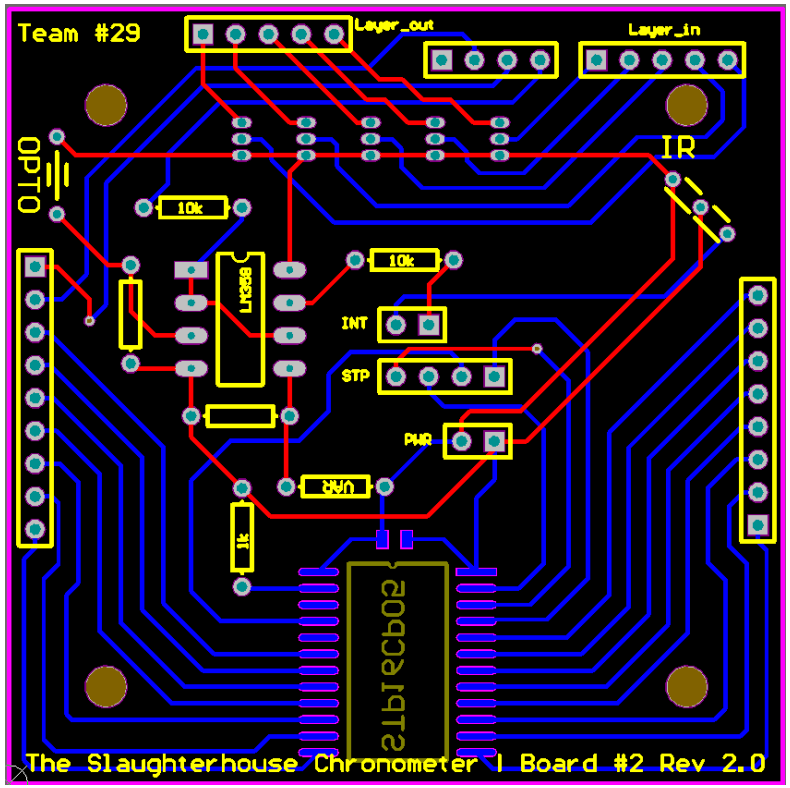


Figure 11: Middle PCB Altium

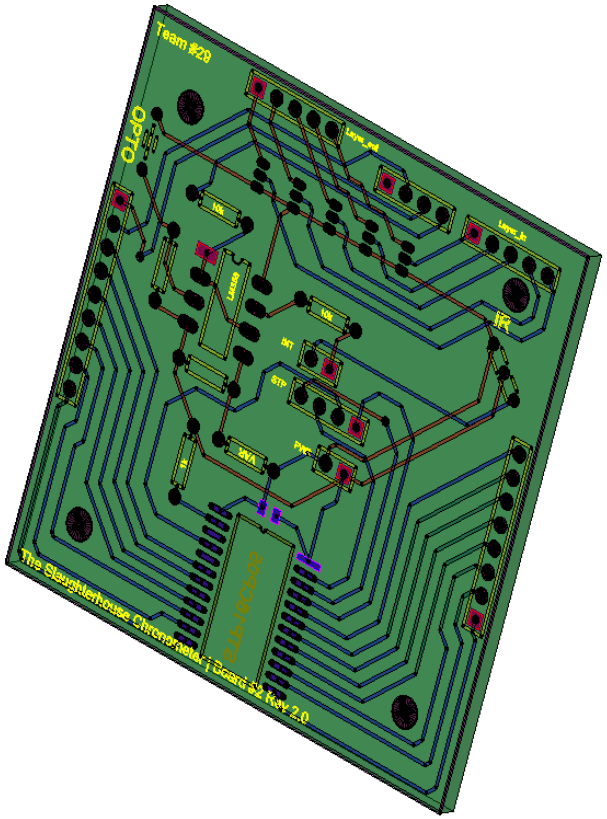


Figure 12: Front PCB 3D Visualisation

4.4.3 PCB Three

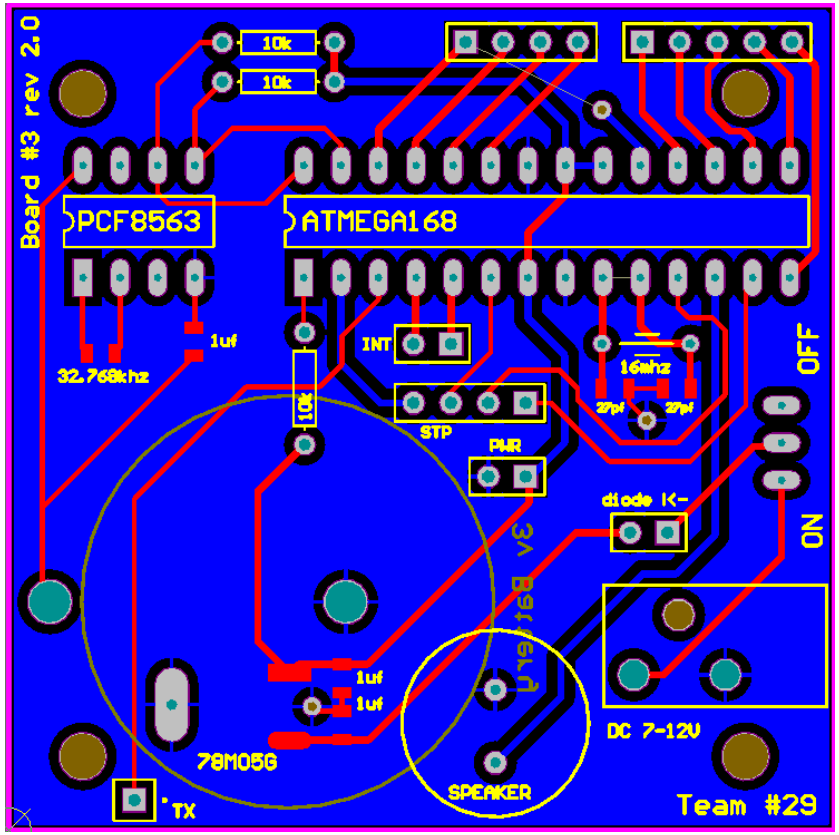


Figure 13: Back PCB Altium

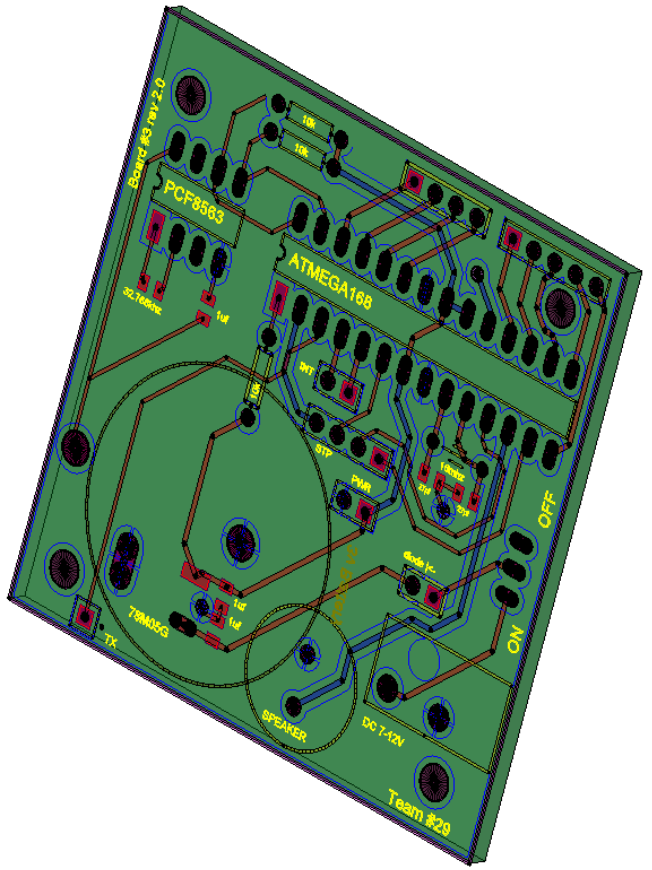


Figure 14: Back PCB 3D Visualisation

4.5 Appendix E: AutoCAD Technical Drawings

Scale: 125mm : 1

4.5.1 Clock Top View

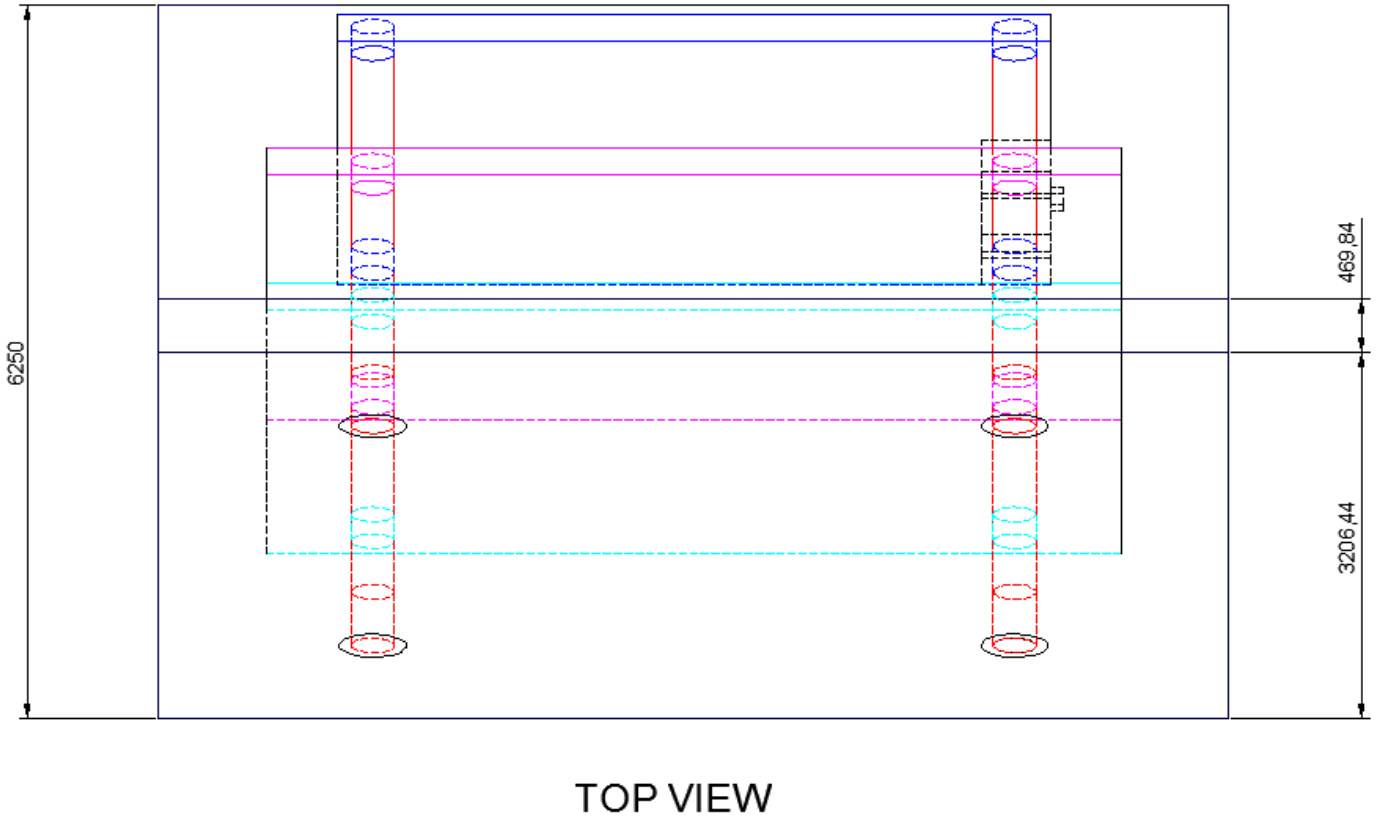


Figure 15: CAD Top View

4.5.2 Clock Front View

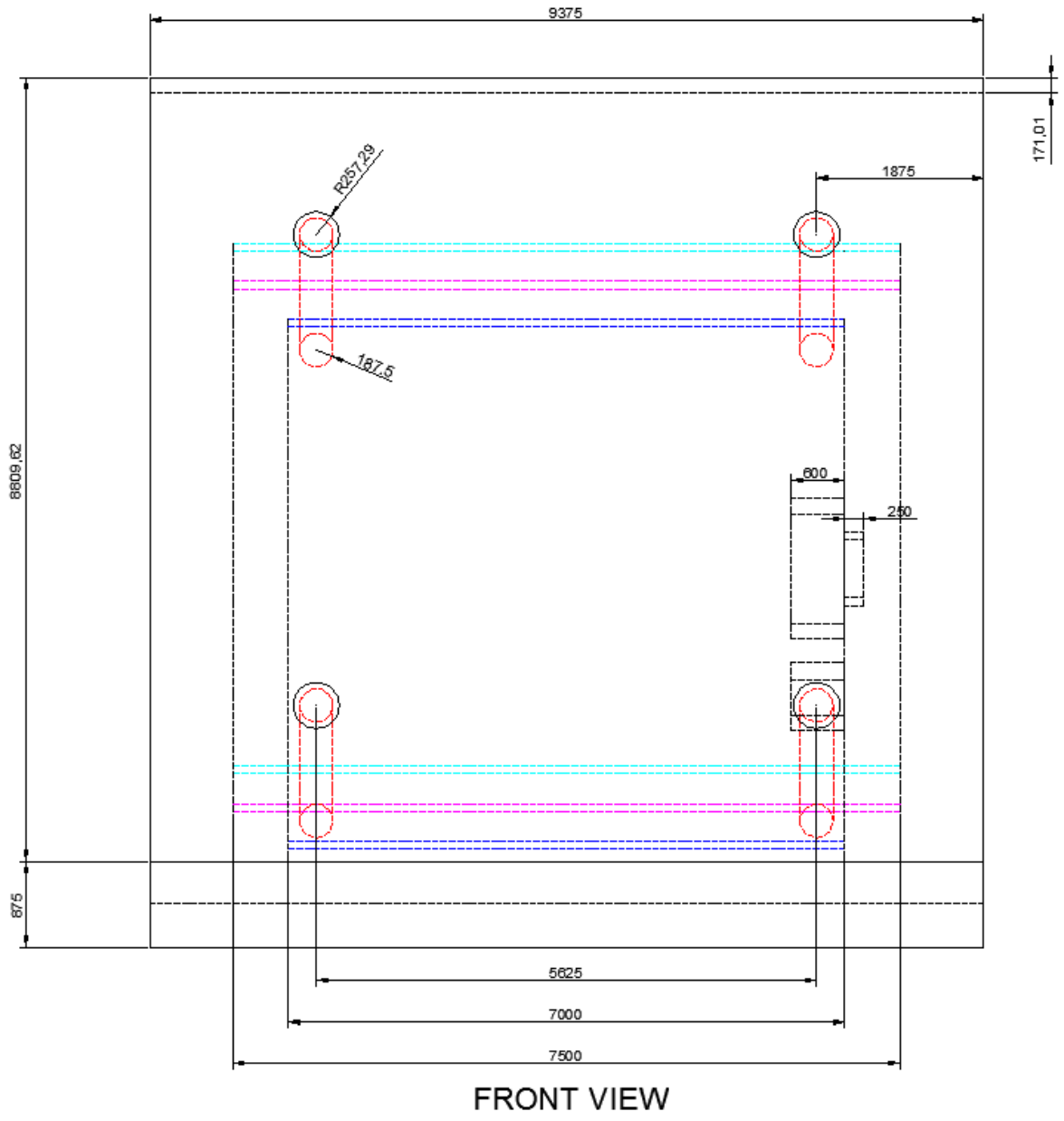


Figure 16: CAD Front View

4.5.3 Clock Side View

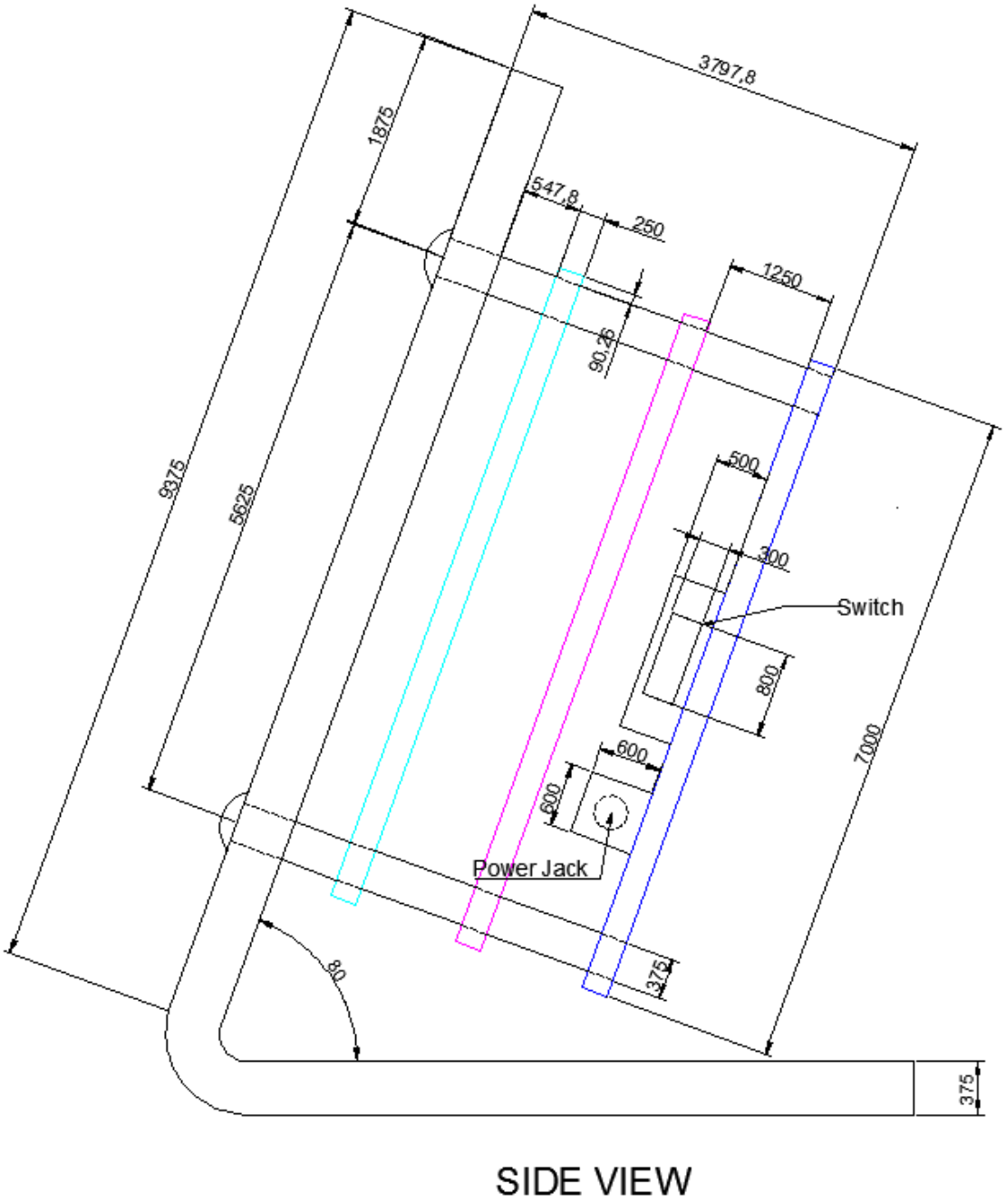
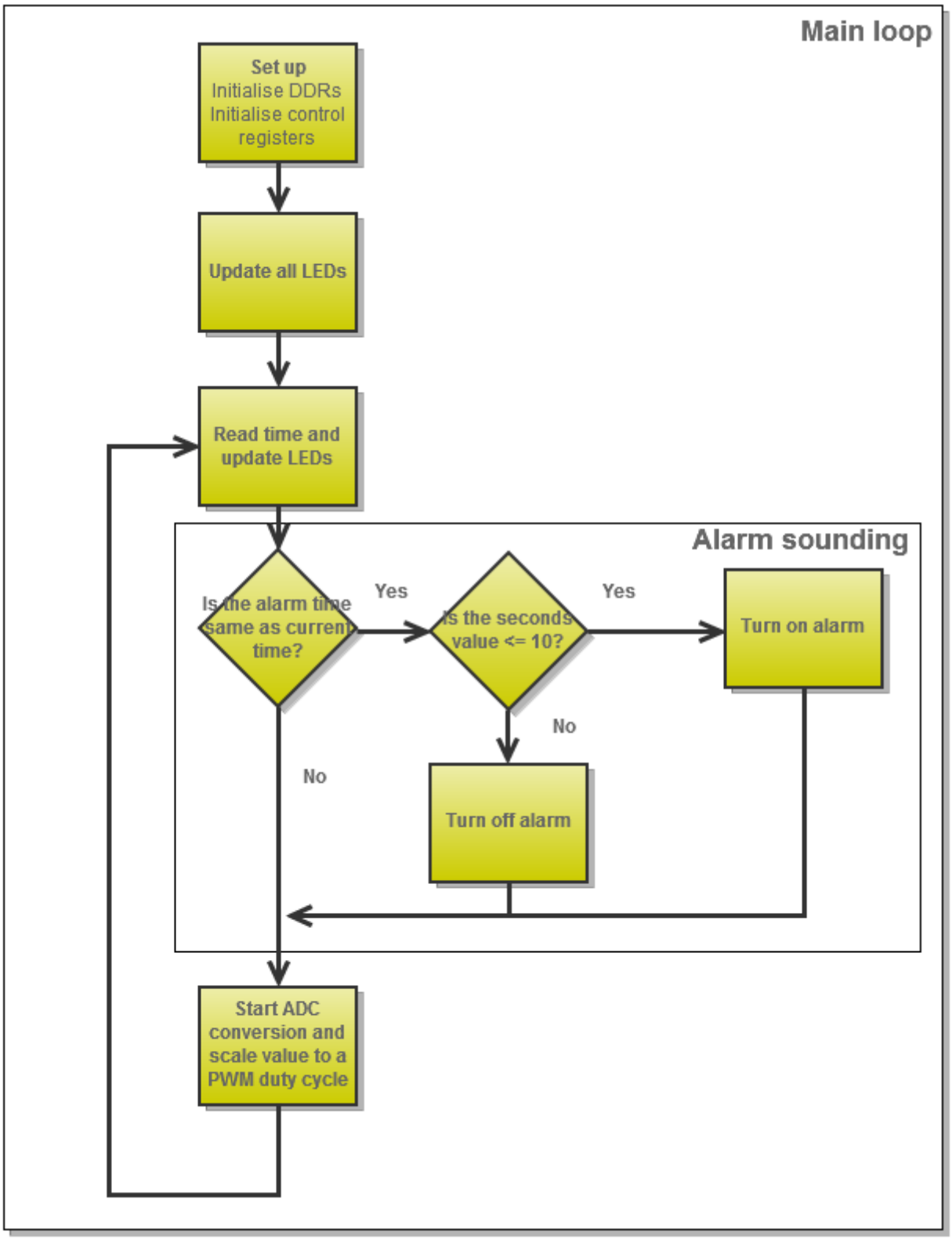


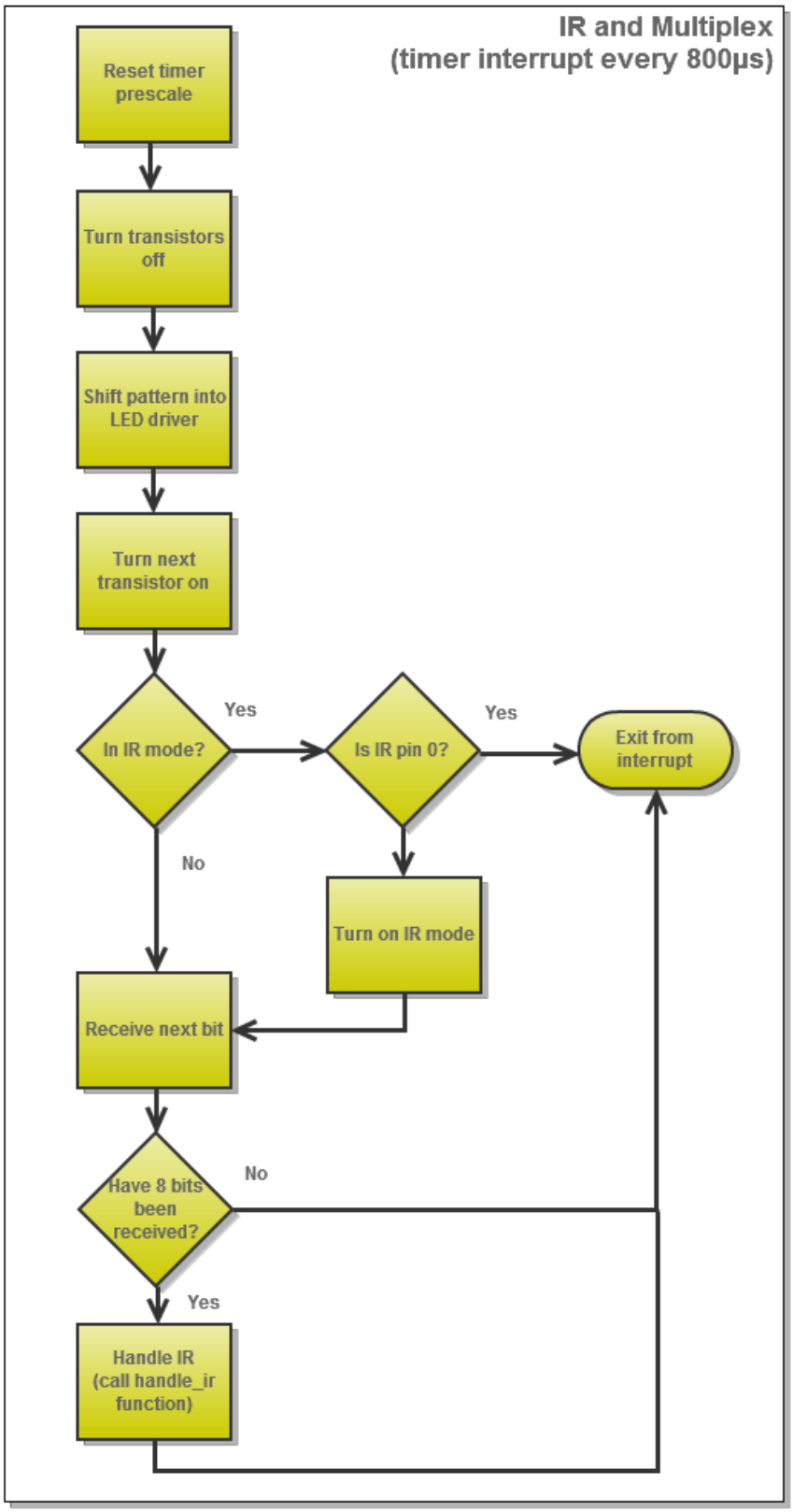
Figure 17: CAD Side View

4.6 Appendix F: Firmware Flowcharts and Code Snippets

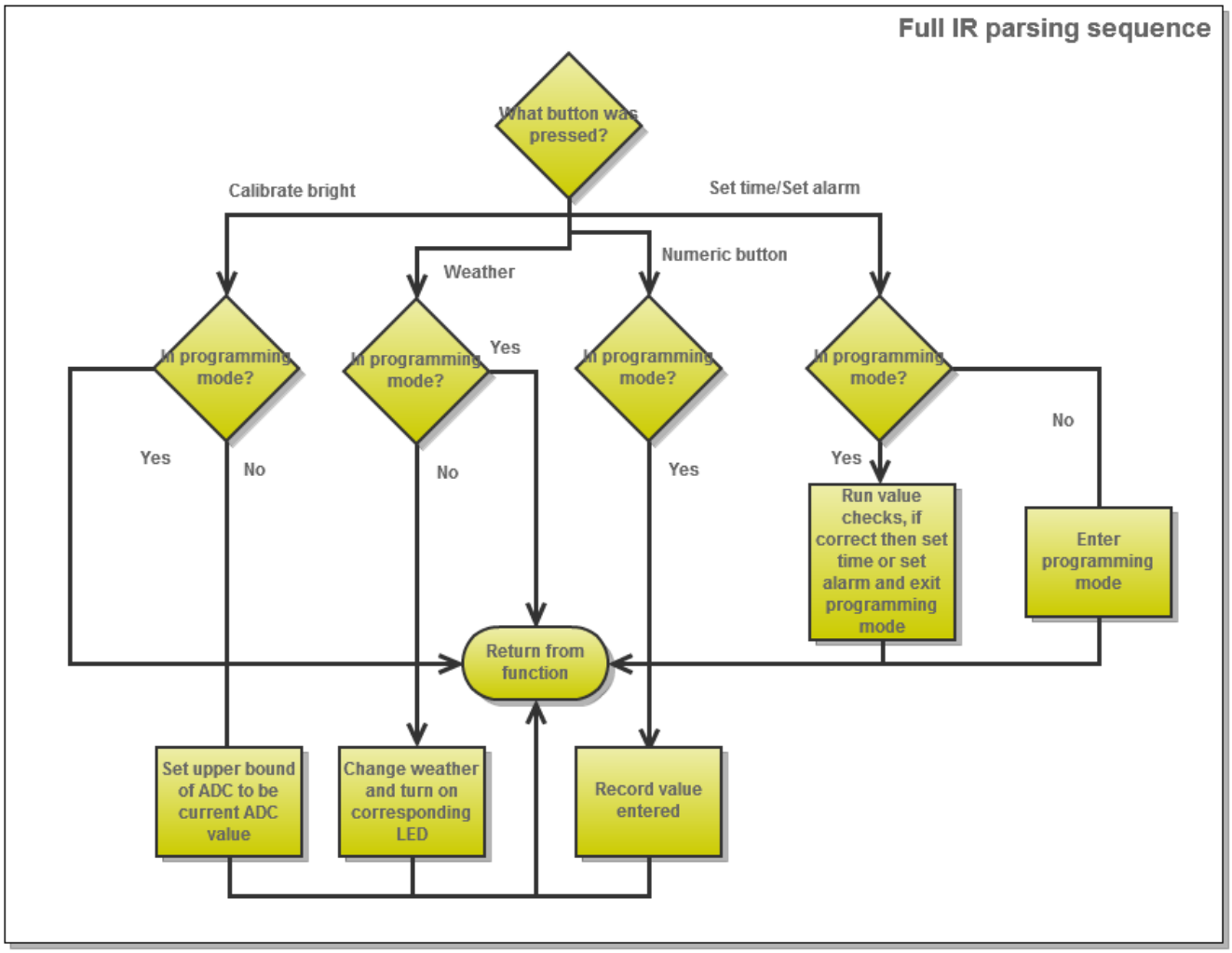
4.6.1 Main loop



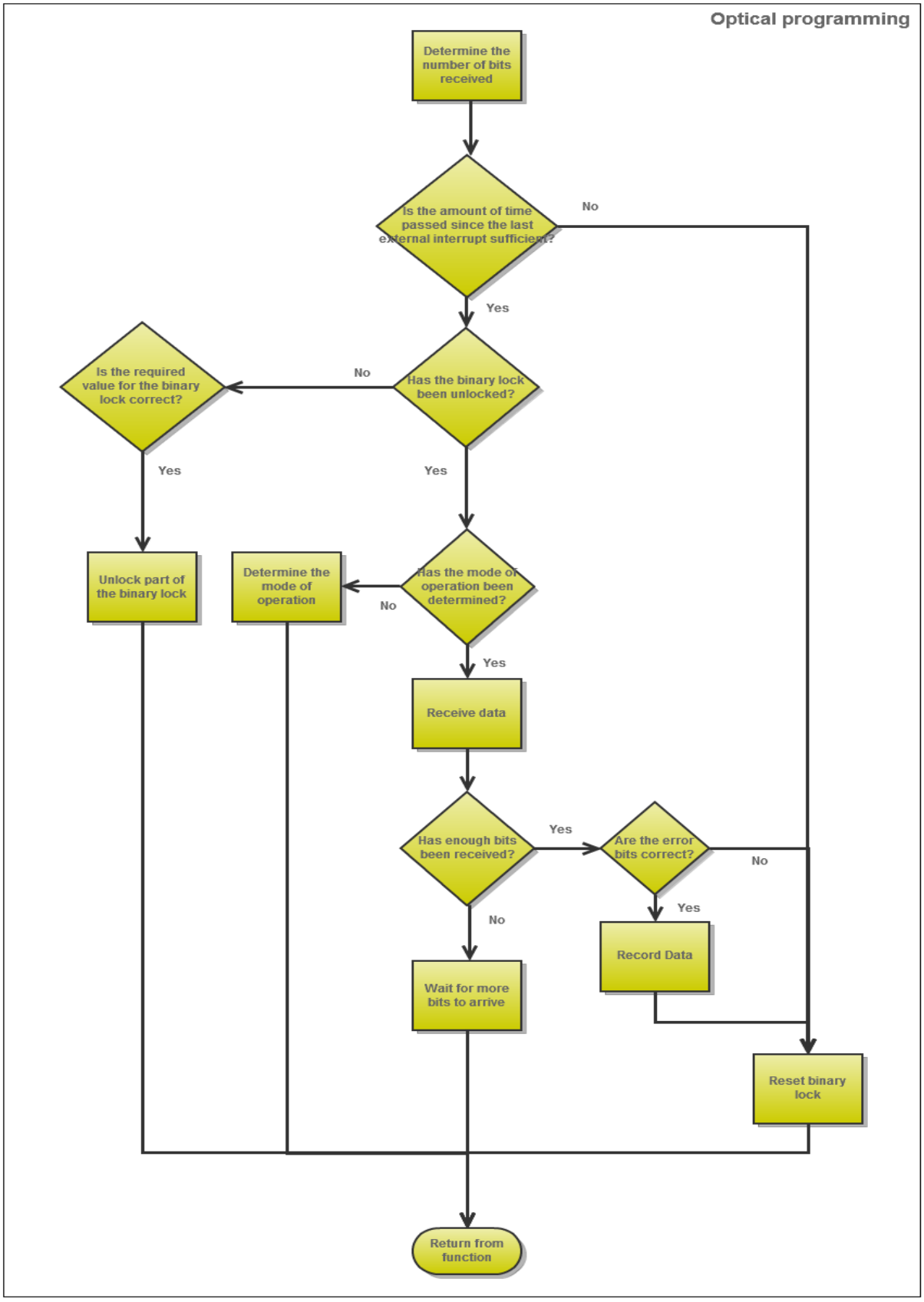
4.6.2 Multiplexing and Infrared programming



4.6.3 Infrared button press parsing (handle_ir function)



4.6.4 Binary Lock Process



4.6.5 Binary Lock Code

```
#define TIMEWAIT 3 // The amount of time that needs to pass for 1 bit to be received
int timeValue = 0; // An integer to indicate how much time has passed
int sequenceNum = 0; // Number of bits that has been unlocked for the binary lock

ISR(TIMER2_COMPA_vect) { // An external interrupt that triggers every 100ms
    timeValue++;
}

ISR(INT1_vect) {
    // Check if the changes in the light sensor are from the PC software
    if((PIND & 0x08) && sequenceNum < 4) {
        // Check if there hasn't been more than one white in the past 300ms
        if ((timeValue / TIMEWAIT) > 1)
            sequenceNum = 1;
        // Check if the value from the light sensor matches the value required
        if (!(sequenceNum % 2))
            sequenceNum++;
        else
            sequenceNum = 1;

    } else if (sequenceNum < 4 && !(PIND & 0x08)) {
        // Check if there hasn't been more than one black in the past 300ms
        if ((timeValue / TIMEWAIT) > 1 && sequenceNum !=3)
            sequenceNum = 0;
        // Check if the value from the light sensor matches the value required
        if (sequenceNum % 2)
            sequenceNum++;
        else
            sequenceNum = 0;
}
```

4.6.6 LED Updating Code

```
/*****************************************************************
 * Generate the correct bit pattern for the current seconds value *
 *****************************************************************/
void update_seconds() {
    // Update the seconds LED buffers
    for (int i = 0; i < 4; i++) {
        // Because buffers are shared, keep current minutes value
        LedPattern currentMin = 0;
        if (time.min % 4 == i)
            currentMin = 1 << (time.min / 4);

        // Compute the pattern to display the current seconds value
        if (time.sec % 4 == i)
            ledPattern[i] = currentMin | (1 << (time.sec / 4));
        else
            ledPattern[i] = currentMin;
    }
}

/*****************************************************************
 * Generate the correct bit pattern for the current minutes value *
 *****************************************************************/
void update_minutes() {
    // Update the minutes LED buffers
    for (int i = 0; i < 4; i++) {
        // Compute the pattern to display the current minutes value
        if (time.min % 4 == i)
            ledPattern[i] = 1 << (time.min / 4);
        else
            ledPattern[i] = 0;
    }

    // When the minutes change, the seconds will always be at the 1st second
    ledPattern[0] |= 1;
}

/***************************************************************
 * Generate the correct bit pattern for the current hours value *
 ***************************************************************/
void update_hours() {
    // Update the hour LED buffer
    ledPattern[4] = 1 << (time.hour);

    update_weather();
}
```

```
/*********************************
 * Set the meridian LED on or off *
 *********************************/
void update_meridian() {
    // Set the MERIDIAN bit to be AM (1) or PM (0)
    if (time.meridian == PM)
        PORTC &= ~(1 << MERIDIAN);
    else
        PORTC |= 1 << MERIDIAN;
}

/************************
 * Set the weather LEDs *
 ************************/
void update_weather() {
    // Clear weather bits
    ledPattern[4] &= ~((1 << CLOUDY) | (1 << RAINY) | (1 << FINE));

    // Update pattern based on weather status
    switch(weather) {
        case NONE:
            break;
        case FINE:
            ledPattern[4] |= 1 << 14;
            break;
        case RAINY:
            ledPattern[4] |= 1 << 13;
            break;
        case CLOUDY:
            ledPattern[4] |= 1 << 12;
    }
}
```

4.6.7 GUI: Button Creation and Event Binding

```
self.time_btn = wx.Button(panel2, pos=(50,50),label='Set Time', size=(110,30)) # Make a button called Set Time
self.time_btn.SetBackgroundColour((242,242,242)) # Set the background colour of the set time button
self.time_btn.SetFont(self.buttonFont) # Set the font of the set time button
self.buttonSizer = wx.BoxSizer(wx.HORIZONTAL) # Create a layout manager for the button
self.buttonSizer.AddSpacer(50) # Add spacing between the edge of the frame to the button
self.buttonSizer.Add(self.time_btn, 0, wx.CENTRE) # Add the button to the GUI
self.Bind(wx.EVT_BUTTON, self.SetTime, self.time_btn) # Bind the method self.SetTime to the button
```

4.6.8 GUI: Setting the Hours

```
hours = str(self.hrs1.GetValue()) + str(self.hrs2.GetValue()) # Grab data from the GUI
hours = bin(int(hours))[2:].zfill(4) # Convert this data to binary
# Send header bits to indicate to the clock to start receiving data
for i in xrange(4):
    if black == True:
        self.flash.SetBackgroundColour(backWhite) # Change the screen to white
        black = False
    else:
        self.flash.SetBackgroundColour(backBlack) # Change the screen to black
        black = True
    self.UpdateDisplay(self.flash) # Update the display
# Send the mode of operation (Setting the time/alarm/weather)
self.flash.SetBackgroundColour(backBlack)
self.UpdateDisplay(self.flash)
self.flash.SetBackgroundColour(backWhite)
self.UpdateDisplay(self.flash)
# Send the hours to the clock
for i in xrange(4):
    if hours[i] == '1': # If the binary bit is one flash white else flash black
        self.flash.SetBackgroundColour(backWhite)
        black = False
    else:
        self.flash.SetBackgroundColour(backBlack)
        black = True
    self.UpdateDisplay(self.flash)
# Send the two bits used for error checking
self.flash.SetBackgroundColour(backBlack)
self.UpdateDisplay(self.flash)
self.flash.SetBackgroundColour(backWhite)
self.UpdateDisplay(self.flash)
```

4.6.9 GUI: Extracting Weather Data

```
weatherResult = pywapi.get_weather_from_yahoo('ASXX0016') # Grab weather data from the server
curCond = weatherResult['condition']['text'] # Extract the necessary weather information
if "fair" in curCond.lower(): # Parse weather information
    weatherSend = 0
elif "cloud" in curCond.lower():
    weatherSend = 1
elif "rain" in curCond.lower():
    weatherSend = 2
else:
    weatherSend = 0
weatherSend = bin(weatherSend)[2:].zfill(2) # Convert weather infromation to binary
```

4.6.10 Main Loop and ADC Code

```
while(1) {
    // Get time from RTC and update LEDs
    read_time();

    // If alarm time is hit, sound alarm for 10 seconds
    if (time.min == alarm.min && time.hour == alarm.hour && time.meridian == alarm.meridian) {
        if (time.sec <= 10)
            OCR0A = (time.sec % 2 ? 100 : 150); // high, low, high, low...
        else
            OCR0A = ALARM_OFF;
    }

    // Autodimming - start reading and wait for ADIF bit to be set
    ADCSRA |= 1 << ADSC;
    while(!bit_is_set(ADCSRA, ADIF));

    // Convert converted value into a duty cycle
    if (ADC < adc_lower)
        pc = adc_lower;
    else if (ADC > adc_upper)
        pc = adc_upper;
    else
        pc = (double) ADC;
    pc = (pc - adc_lower) / (adc_upper - adc_lower);

    // Set new duty cycle
    OCR0B = (int)(MIN_DUTY_CYCLE * (1 - pc));
}
```